Experiences Utilizing CPUs and GPUs for Computation Simultaneously on a Heterogeneous Node

COEPP 2017
August 22-24, 2017

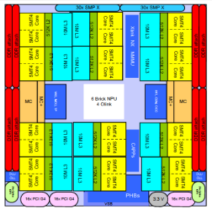
Lawrence Livermore National Laboratory

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract. DE-AC52-07NA27344.

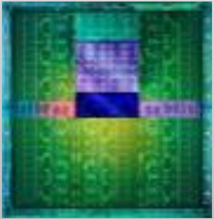
Olga Pearce
Lawrence Livermore National Laboratory
http://people.llnl.gov/olga

Sierra node is a POWER9 2 Socket Server

2x POWER9



4x Volta



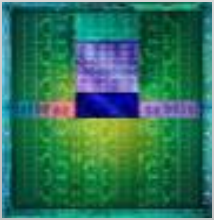
Sierra node is a POWER9 2 Socket Server

2x POWER9

30x SMP X
30x SMP X
SMT4 Core
512K L2
10M L3
MC
6 Brick NPU
4 Olink
PHBs
3.3 V
16x PCI G4
16x PCI G4
16x PCI G4

5%
FLOPS

4x Volta

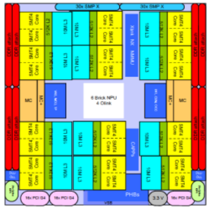


95%
FLOPS

- Use the GPUs effectively

Sierra node is a POWER9 2 Socket Server

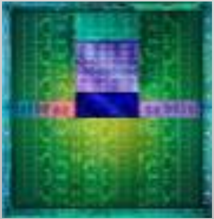
2x POWER9



5%
FLOPS

- Use the CPUs as well

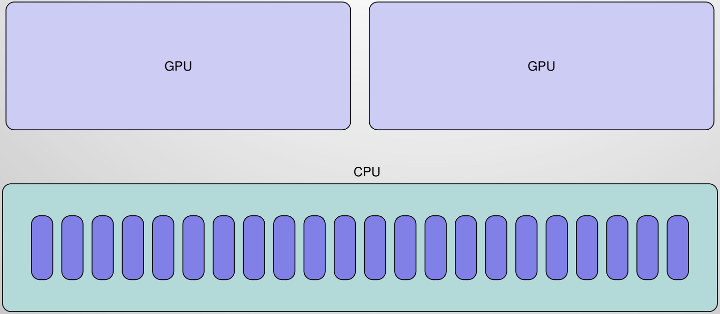
4x Volta



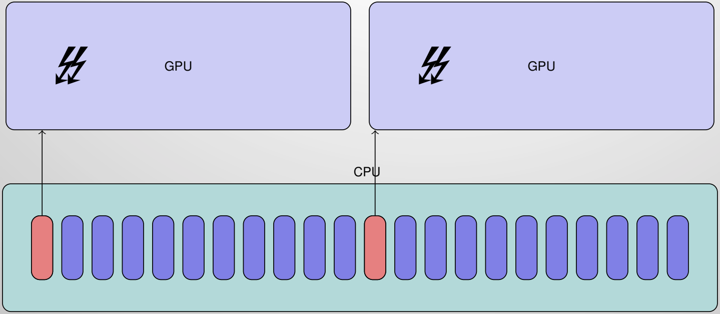
95%
FLOPS

- Use the GPUs effectively

Ways to utilize a node of Sierra (showing one socket)

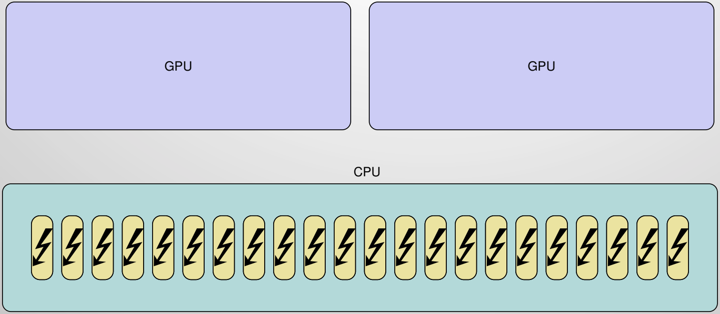


Ways to utilize a node of Sierra (showing one socket)



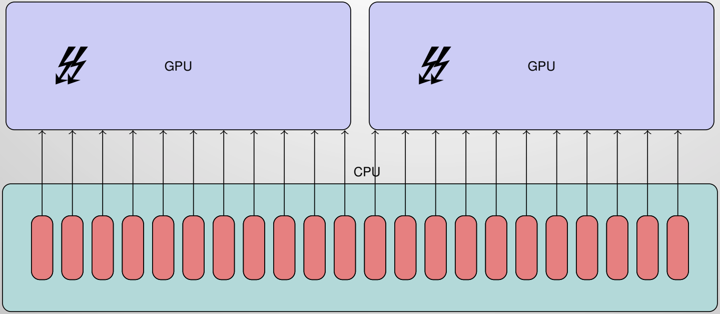
1. Single MPI task per GPU

Ways to utilize a node of Sierra (showing one socket)



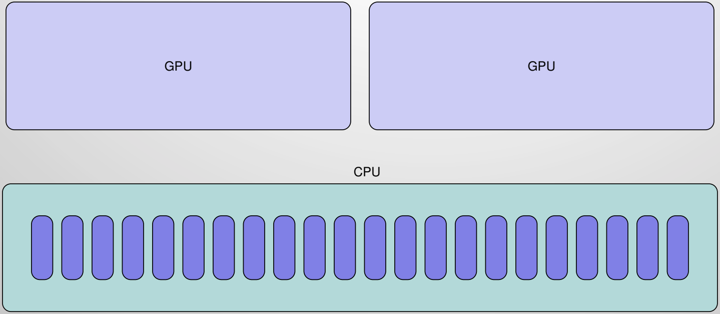
1. Single MPI task per GPU
2. Single MPI task per core

Ways to utilize a node of Sierra (showing one socket)



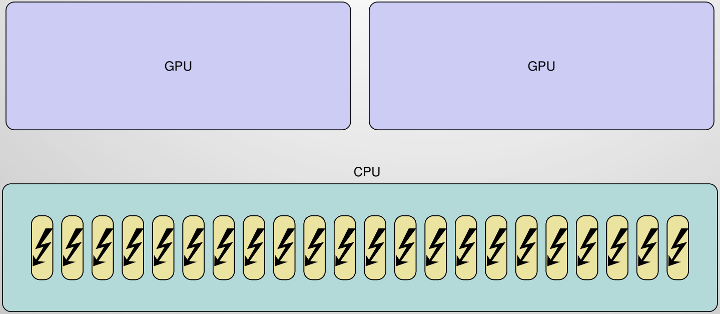
1. Single MPI task per GPU
2. Single MPI task per core
   - Multiple MPI tasks per GPU with Multi-Process Service (MPS)

Ways to utilize a node of Sierra (showing one socket)



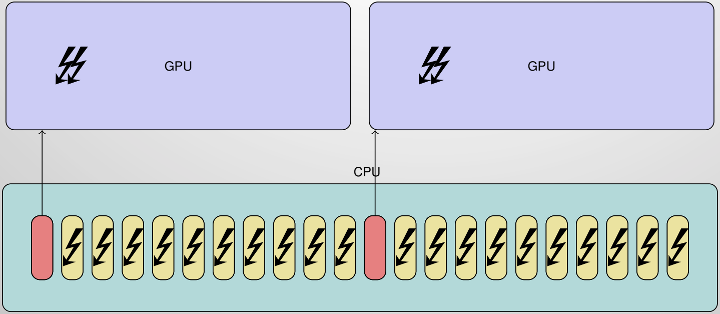
1. Single MPI task per GPU
2. Single MPI task per core
   - Multiple MPI tasks per GPU with Multi-Process Service (MPS)
3. Heterogeneous MPI tasks

Ways to utilize a node of Sierra (showing one socket)



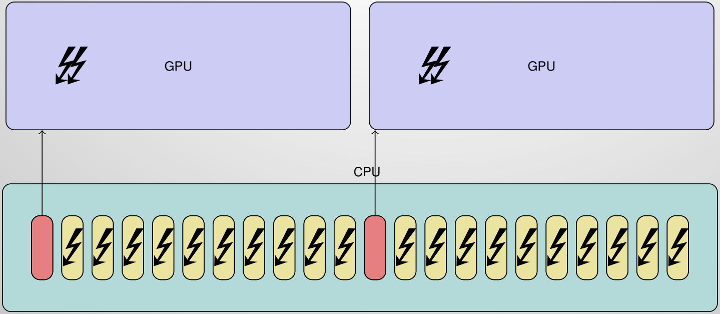
1. Single MPI task per GPU
2. Single MPI task per core
    - Multiple MPI tasks per GPU with Multi-Process Service (MPS)
3. Heterogeneous MPI tasks
    - Single MPI task per core

Ways to utilize a node of Sierra (showing one socket)



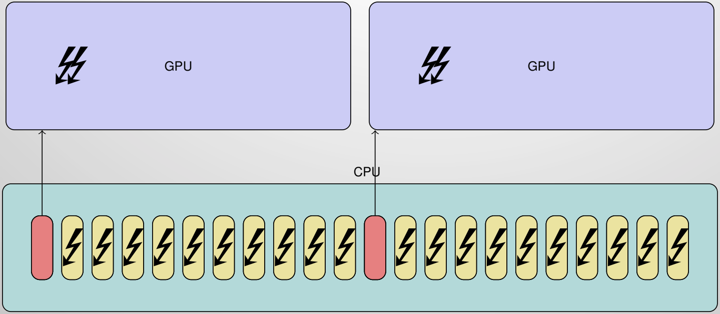
1. Single MPI task per GPU
2. Single MPI task per core
   - Multiple MPI tasks per GPU with Multi-Process Service (MPS)
3. Heterogeneous MPI tasks
   - Single MPI task per core
   - Some compute, some ‘drive’ the GPU

Ways to utilize a node of Sierra (showing one socket)



1. Single MPI task per GPU ← launching big kernels?
2. Single MPI task per core ← launching many small kernels?
   - Multiple MPI tasks per GPU with Multi-Process Service (MPS)
3. Heterogeneous MPI tasks ← when 1 core sufficient to drive GPU?
   - Single MPI task per core
   - Some compute, some ‘drive’ the GPU

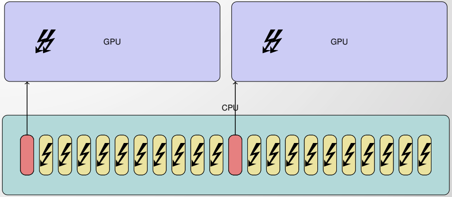
Ways to utilize a node of Sierra (showing one socket)



1. Single MPI task per GPU ← launching big kernels?
2. Single MPI task per core ← launching many small kernels?
   - Multiple MPI tasks per GPU with Multi-Process Service (MPS)
3. Heterogeneous MPI tasks ← when 1 core sufficient to drive GPU?
   - Single MPI task per core
   - Some compute, some ‘drive’ the GPU

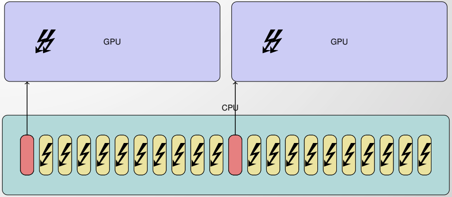
⇒ Hard to project performance to future hardware

Heterogeneous MPI tasks



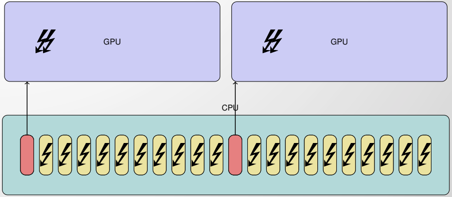
1. Control code
   - Some MPI processes ‘drive’ the GPUs
   - Other MPI processes compute on the CPU cores
   - Be careful about the CPU core/GPU binding!

Heterogeneous MPI tasks



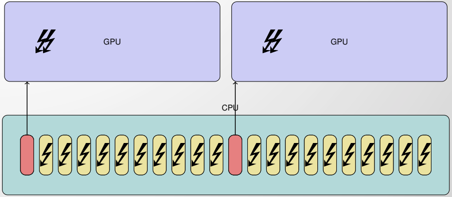
1. Control code
   - Some MPI processes ‘drive’ the GPUs
   - Other MPI processes compute on the CPU cores
   - Be careful about the CPU core/GPU binding!
2. Memory allocation
   - Allocate the memory according to the task of the particular CPU core
   - This was fairly straight-forward for a spatially decomposed MPI application where each MPI process owns its data

Heterogeneous MPI tasks



1. Control code
   - Some MPI processes ‘drive’ the GPUs
   - Other MPI processes compute on the CPU cores
   - Be careful about the CPU core/GPU binding!
2. Memory allocation
   - Allocate the memory according to the task of the particular CPU core
   - This was fairly straight-forward for a spatially decomposed MPI application where each MPI process owns its data
3. Loop execution
   - Portability: same source code?
   - Proof of concept implementation in ARES using RAJA

Heterogeneous MPI tasks

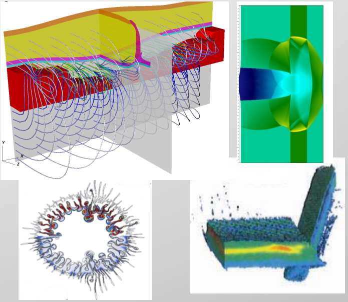


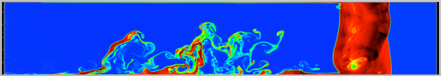
1. Control code
   - Some MPI processes ‘drive’ the GPUs
   - Other MPI processes compute on the CPU cores
   - Be careful about the CPU core/GPU binding!
2. Memory allocation
   - Allocate the memory according to the task of the particular CPU core
   - This was fairly straight-forward for a spatially decomposed MPI application where each MPI process owns its data
3. Loop execution
   - Portability: same source code?
   - Proof of concept implementation in ARES using RAJA
4. Communication
   - Haven’t explored GPU direct yet

ARES is a massively parallel, multi-dimensional, multi-physics code [from Brian Ryujin's slides]

Physics Capabilities:

- ALE-AMR Hydrodynamics
- High-order Eulerian Hydrodynamics
- Elastic-Plastic flow
- 3T plasma physics
- High-Explosive modeling
- Diffusion, S_N Radiation
- Particulate flow
- Laser ray-tracing
- MHD
- Dynamic mixing
- Non-LTE opacities





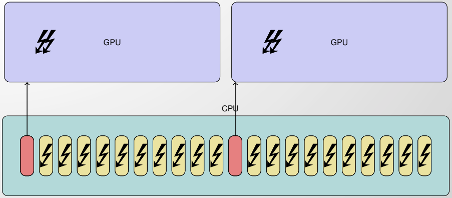
Applications:

- ICF modeling
- Pulsed power
- NIF Debris
- High-Explosive experiments

memory allocation in ARES

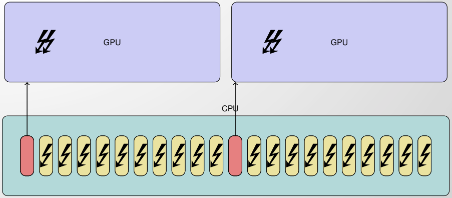
- Differentiate memory use by context
  - Malloc - CPU control code
  - cudaMallocManaged(UM) - mesh data (accessed on CPU & GPU)
  - cudaMalloc (cnmem memory pools) - Temporary GPU data

Heterogeneous memory allocation in ARES



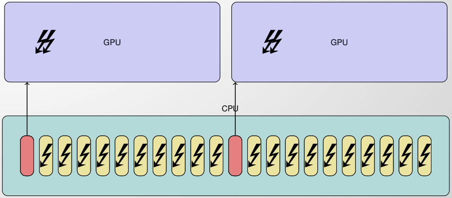
- Differentiate memory use by context
  - Malloc - CPU control code
  - cudaMallocManaged(UM) - mesh data (accessed on CPU & GPU)
  - cudaMalloc (cnmem memory pools) - Temporary GPU data

Heterogeneous memory allocation in ARES



- Differentiate memory use by context
  - Malloc - CPU control code
  - cudaMallocManaged(UM) - mesh data (accessed on CPU & GPU)
  - cudaMalloc (cnmem memory pools) - Temporary GPU data
- Use control code to inject additional context
  - If the MPI process is ‘driving’ the GPU, do the above
  - If the MPI process is executing loops on the CPU core, allocate everything on the CPU

Heterogeneous memory allocation in ARES



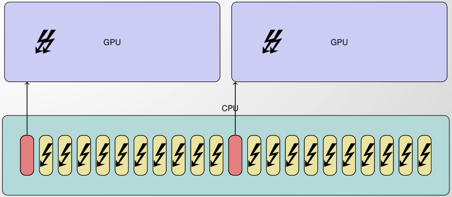
- Differentiate memory use by context
  - Malloc - CPU control code
  - cudaMallocManaged(UM) - mesh data (accessed on CPU & GPU)
  - cudaMalloc (cnmem memory pools) - Temporary GPU data
- Use control code to inject additional context
  - If the MPI process is ‘driving’ the GPU, do the above
  - If the MPI process is executing loops on the CPU core, allocate everything on the CPU
- Gotchas
  - Dependencies may assume that ‘USE_CUDA’ == allocate on GPUs
  - Touching UM from the CPU-only MPI process will slow things down

loop execution in ARES

```
RAJA::forall<AresPolicy>(. . . , kernel);
```

- AresPolicy: not thread safe, thread safe, etc.

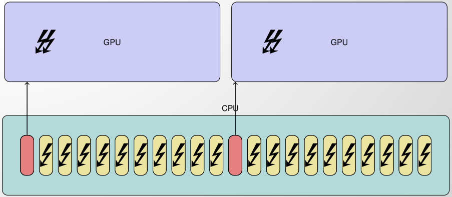
Heterogeneous loop execution in ARES



1: **RAJA::forall**<AresPolicy>(. . . , kernel);

- AresPolicy: not thread safe, thread safe, etc.
- AresArchPolicy: At runtime, select the appropriate policy according to where want to execute (future: MultiPolicy in RAJA)

Heterogeneous loop execution in ARES



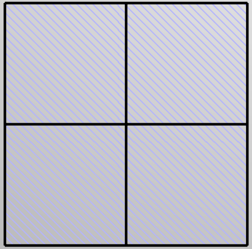
```
RAJA::forall<AresPolicy>(. . . , kernel);
```

- AresPolicy: not thread safe, thread safe, etc.
- AresArchPolicy: At runtime, select the appropriate policy according to where want to execute (future: MultiPolicy in RAJA)

```
if (run_on_gpu) then
    //RAJA backend: GPU specific (CUDA, OpenMP)
    typedef DynamicPolicy<AresPolicy, GPU> AresArchPolicy;
    RAJA::forall<AresArchPolicy>(. . . , kernel);
else
    //RAJA backend: CPU specific (Serial, OpenMP)
    typedef DynamicPolicy<AresPolicy, CPU> AresArchPolicy;
    RAJA::forall<AresArchPolicy>(. . . , kernel);
end if
```

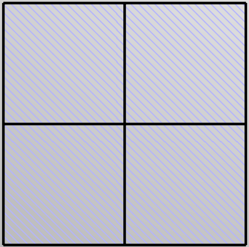
domain decomposition in ARES

1MPI/GPU

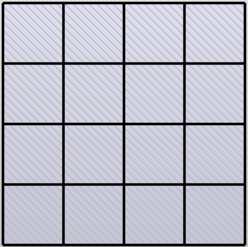


domain decomposition in ARES

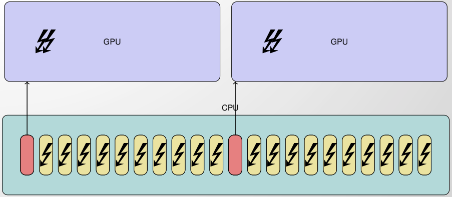
1MPI/GPU

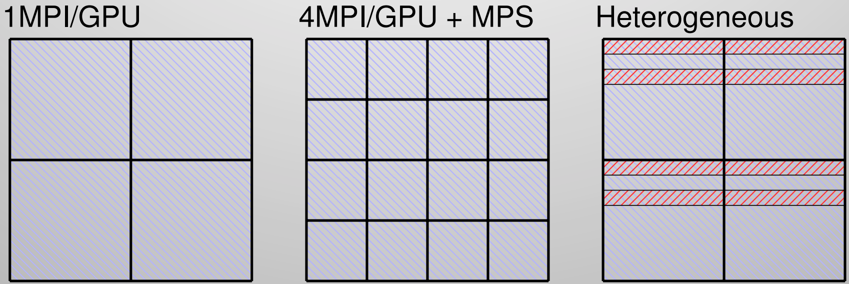


4MPI/GPU + MPS

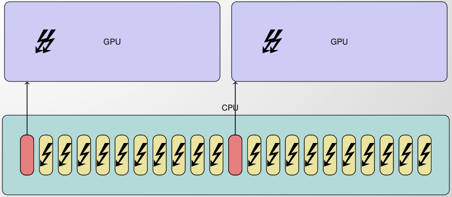


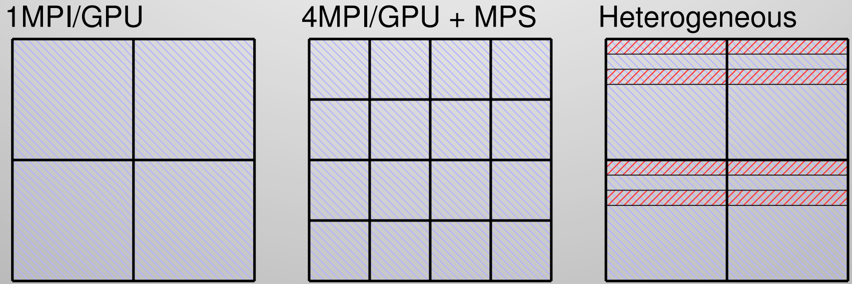
Heterogeneous domain decomposition in ARES





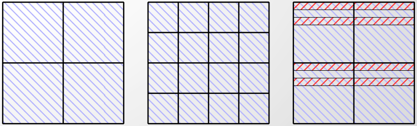
Heterogeneous domain decomposition in ARES





- Use hierarchical decomposition for Heterogeneous approach
- Decomposition impacts memory accesses

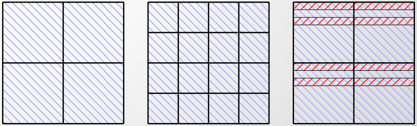
Performance comparison



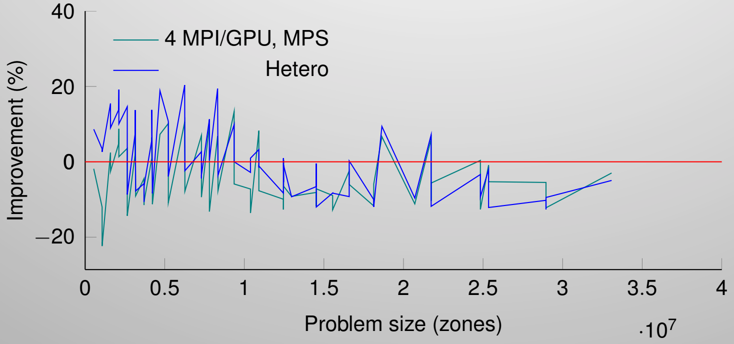
- ARES, 3D Sedov problem
- rzhasgpu, CUDA RAJA backend[1]
- Baseline: 1MPI/GPU

[1] All results generated with pre-release versions of IBM compilers; improvements in performance expected in future releases

Performance comparison

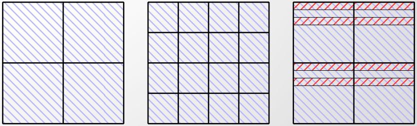


- ARES, 3D Sedov problem
- rzhasgpu, CUDA RAJA backend[1]
- Baseline: 1MPI/GPU

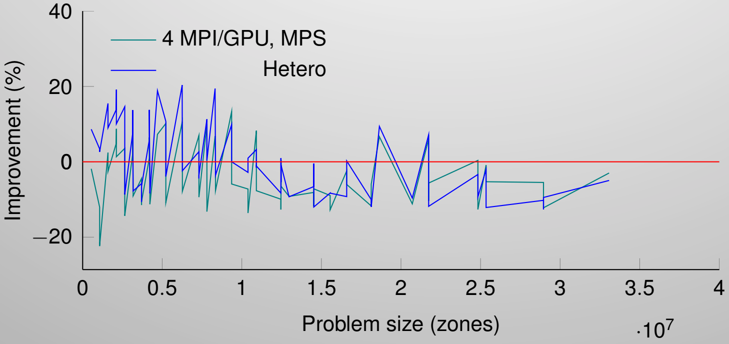


[1] All results generated with pre-release versions of IBM compilers; improvements in performance expected in future releases

Performance comparison



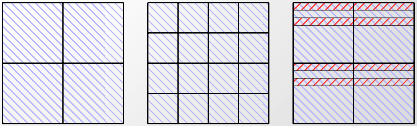
- ARES, 3D Sedov problem
- rzhasgpu, CUDA RAJA backend[1]
- Baseline: 1MPI/GPU

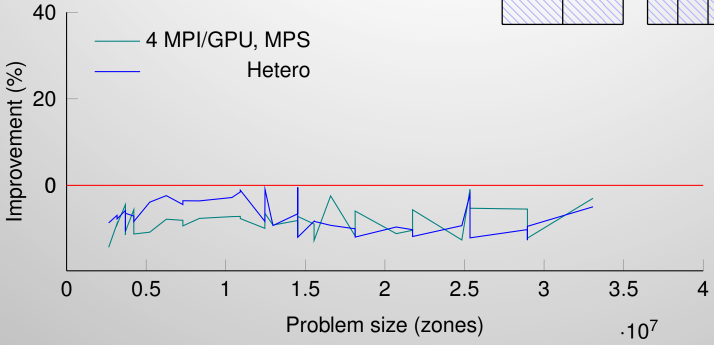


- Size of inner-loop dimension impacts performance

[1] All results generated with pre-release versions of IBM compilers; improvements in performance expected in future releases

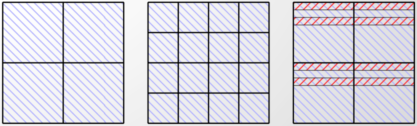
Best: 1MPI/GPU

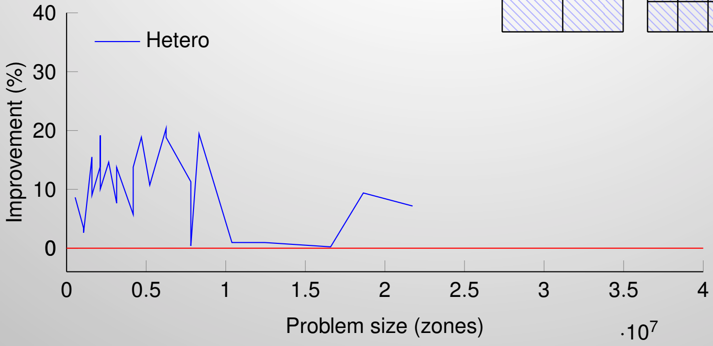




- Baseline: 1MPI/GPU
- When inner-loop dimension is large
  - Memory use is optimal with 1MPI/GPU (4MPI/GPU results in smaller inner-loop dimension)
  - Can’t take a small enough chunk of work to give to the CPU

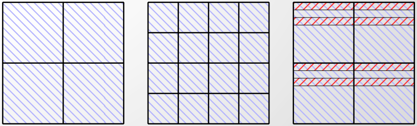
Best: Heterogeneous

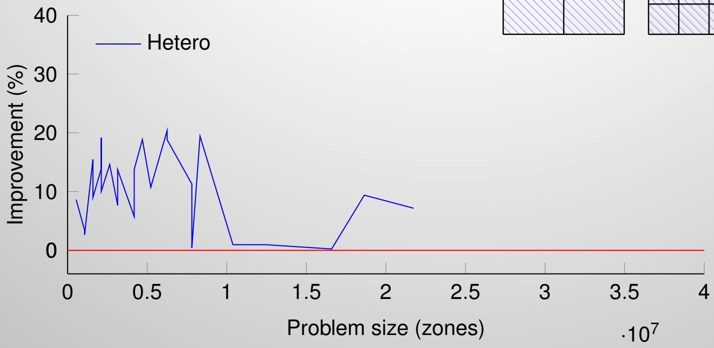




- Memory use the same as for 1MPI/GPU (slice in Y dimension)
- When Y dim. is large, can give smaller portions of work to CPU

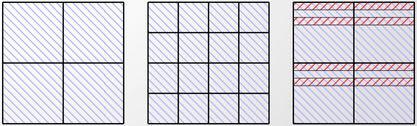
Best: Heterogeneous

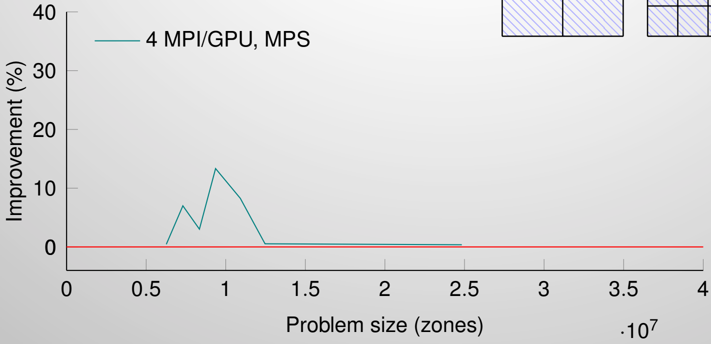




- Memory use the same as for 1MPI/GPU (slice in Y dimension)
- When Y dim. is large, can give smaller portions of work to CPU
- Right now, only give 1-2% of work to the CPU
  - __host__ __device__ decorated lambdas are significantly slower when running on the CPU because nvcc passes the lambda back to the host compiler wrapped in a std::function object.

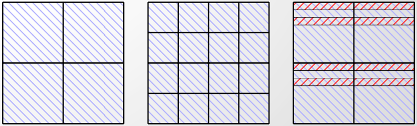
Best: 4MPI/GPU+MPS

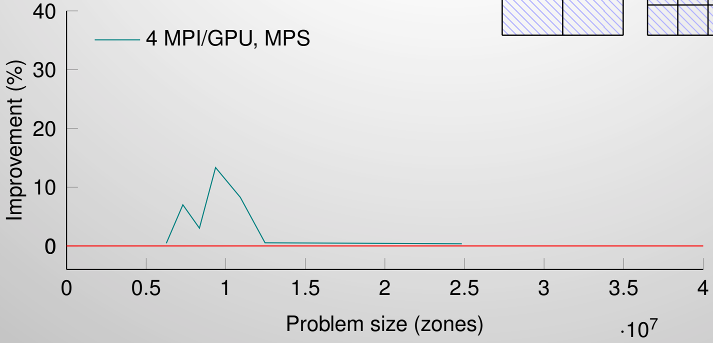




- In all of these cases, ARES decomposition didn't cut further in X dimensions because Y and Z dimensions were large
- Memory use the same as for 1MPI/GPU

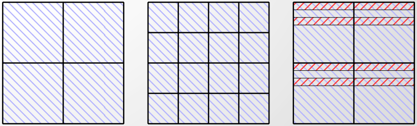
Best: 4MPI/GPU+MPS

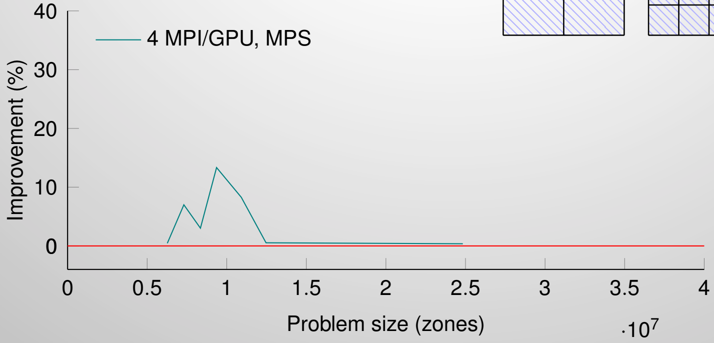




- In all of these cases, ARES decomposition didn't cut further in X dimensions because Y and Z dimensions were large
- Memory use the same as for 1MPI/GPU
- MPS may be beneficial if we use a special hierarchical decomposition

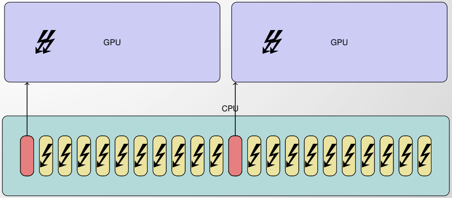
Best: 4MPI/GPU+MPS





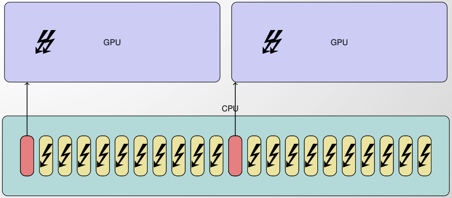
- In all of these cases, ARES decomposition didn't cut further in X dimensions because Y and Z dimensions were large
- Memory use the same as for 1MPI/GPU
- MPS may be beneficial if we use a special hierarchical decomposition
- Performance with MPS keeps changing - keep reevaluating

Heterogeneous load balancing



- ‘Direction’ of split certainly impacts memory performance
- Have to take into consideration memory access overhead, data transfer overhead, kernel launch overhead, etc.

Heterogeneous load balancing

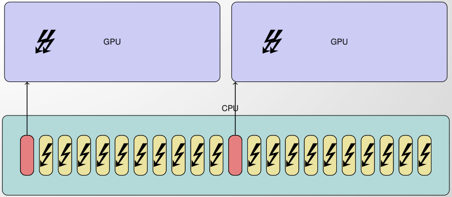


- ‘Direction’ of split certainly impacts memory performance
- Have to take into consideration memory access overhead, data transfer overhead, kernel launch overhead, etc.

Related work:
Heterogeneous task scheduling for accelerated OpenMP. Thomas R.W. Scogland, Barry Rountree, Wu-Chun Feng, Bronis R. de Supinski. Parallel & Distributed Processing Symposium (IPDPS), May 2012.

- Proposed changes to OpenMP which allow task scheduling on both the CPU and GPU
- Calculated the ratio for splitting the iterations via a linear program

Conclusions



- Proof of concept implementation for utilizing the GPUs and all CPU cores to perform loop computation in ARES
- Performance portability courtesy of RAJA (same source code for CPU and GPU)
- Divide work via domain decomposition
- Load balancing between the CPUs and GPUs is non-trivial
- Compared performance of the 1MPI/GPU implementation, heterogeneous implementation, and 4MPI/GPU+MPS
- Performance with MPS is likely to change
- Memory access pattern dominates performance
- ‘Square’ domains may no longer be optimal